DEVIGNE Lisa
EMEkLI Ese
Date du TP (06/12/2021)

Rapport de TP3 : Représentation visuelle d'objets.

I. Introduction

On s'intéresse dans ce TP à la représentation d'objets 3D à l'écran dans une fenêtre graphique qui permet des opérations de zoom, rotation et translations. On a choisi la représentation de maisons à partir d'objets simples que l'on va construire progressivement comme les murs, les portes, les fenêtres…

II. Préparation à faire avant TP – Utilisation de Pygame

1. Question (1)

```
import pygame
pygame.init()
ecran = pygame.display.set_mode((300, 200))
pygame.quit()
```

Explications du code :
On importe d'abord la bibliothèque pygame dans spyder. Ensuite, on initialise tous les modules de pygame importés. On règle ensuite les dimensions (300x200) de la surface d'affichage et puis on quitte pygame.

Lorsqu'on teste le code, on a une fenêtre pygame window qui s'ouvre puis qui se ferme rapidement.

2. Question (2)

```
import pygame

pygame.init()
ecran = pygame.display.set_mode((300, 200))

continuer = True
while continuer:
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            continuer = False

pygame.quit()
```

Explication du code :
On importe d'abord la bibliothèque pygame dans spyder. Ensuite, on initialise tous les modules de pygame importés. On règle ensuite les dimensions (300x200) de la surface d'affichage et puis on quitte pygame. La fenêtre pygame window ne se ferme pas tant qu'il n'y pas « d'évènement », c'est-à-dire une touche du clavier appuyée. Lorsque l'utilisateur appuie sur une touche, cette fenêtre se ferme.

Lorsqu'on teste le code, on a une fenêtre pygame window qui s'ouvre puis on appui sur n'importe quelle touche et la fenêtre se ferme.

III. Préparation à faire avant TP – Utilisation de PyopenGl pour représenter des objets 3D

1. Question (1)

Aspect correspond au rapport de la hauteur par la largeur. Nous avons donc divisé display [0] par display [1].
Nous avons utilisé la fonction gluPerspective : *gluPerspective (45, display [0]/display [1], 0.1, 50)*

2. Question (2)

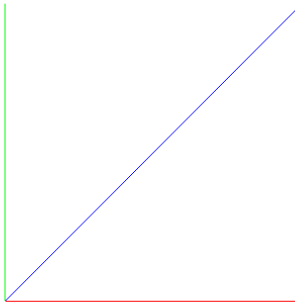
RGB correspond aux couleurs Red Green Blue.
L'axe x est représenté en rouge par (255, 0, 0)
L'axe y est représenté en vert par (0, 255, 0)
L'axe Z est représenté en bleu par (0, 0, 255)

Voici notre code avec son affichage : l'axe z est en profondeur

```
import pygame
import OpenGL.GL as gl
import OpenGL.GLU as glu

if __name__ == '__main__':
    pygame.init()
    display=(600,600)
    pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL)

      # Sets the screen color (white)
    gl.glClearColor(1, 1, 1, 1)
    # Clears the buffers and sets DEPTH_TEST to remove hidden surfaces
    gl.glClear(gl.GL_COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT)
    gl.glEnable(gl.GL_DEPTH_TEST)
    glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
    gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode lignes (segments)

    gl.glColor3fv([255, 0, 0]) # Indique la couleur du prochian segment en RGB
    gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
    gl.glVertex3fv((1, 0, -2)) # Deuxième vertice : fin de la ligne


    gl.glColor3fv([0, 255, 0]) # Indique la couleur du prochian segment en RGB
    gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
    gl.glVertex3fv((0, 1, -2)) # Deuxième vertice : fin de la ligne



    gl.glColor3fv([0, 0, 255]) # Indique la couleur du prochian segment en RGB
    gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
    gl.glVertex3fv((0, 0, -1)) # Deuxième vertice : fin de la ligne

    gl.glEnd() # Find du tracé
    pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique



    # Placer ici l'utilisation de gluPerspective.


    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
    exit()
```

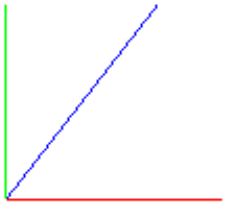
On modifie les coordonnés de la 2e vertice du 3ème tracé pour pouvoir visualiser l'affichage de l'axe Z :

```
import pygame
import OpenGL.GL as gl
import OpenGL.GLU as glu

if __name__ == '__main__':
    pygame.init()
    display=(600,600)
    pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL)

      # Sets the screen color (white)
    gl.glClearColor(1, 1, 1, 1)
    # Clears the buffers and sets DEPTH_TEST to remove hidden surfaces
    gl.glClear(gl.GL_COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT)
    gl.glEnable(gl.GL_DEPTH_TEST)
    glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
    gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode lignes (segments)

    gl.glColor3fv([255, 0, 0]) # Indique la couleur du prochian segment en RGB
    gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
    gl.glVertex3fv((1, 0, -2)) # Deuxième vertice : fin de la ligne


    gl.glColor3fv([0, 255, 0]) # Indique la couleur du prochian segment en RGB
    gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
    gl.glVertex3fv((0, 1, -2)) # Deuxième vertice : fin de la ligne



    gl.glColor3fv([0, 0, 255]) # Indique la couleur du prochian segment en RGB
    gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
    gl.glVertex3fv((1, 1, -1)) # Deuxième vertice : fin de la ligne

    gl.glEnd() # Find du tracé
    pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique



    # Placer ici l'utilisation de gluPerspective.


    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
    exit()
```

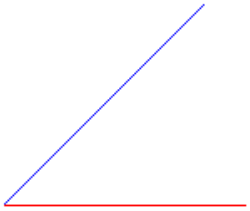


3. Question (3)

Translation de z= -5 :



Puis rotation autour axe x de 90° :



IV. Préparation à faire avant TP - Découverte de l'environnement de travail du TP

1. Question (1) a

La touche a permet de faire disparaître puis réapparaître le tracé.
La touche z permet de faire une rotation dans le sens trigonométrique.
La touche Z permet de faire une rotation dans le sens horaire.

Fichier Configuration.py :
Il crée l'environnement d'affichage. Il définit une configuration par 3 axes (x,y,z) ainsi que leur couleurs et la position de l'écran. Il comporte une fonction qui génère les coordonnées des axes, une qui permet d'ajouter des objets (de type porte, murs etc.) dans une Configuration et également une fonction qui dessine les axes et les différents

objets. Il y a aussi des fonctions qui définissent les évènement (actions de l'utiliateur) comme KeyDown, MouseButtonDown, et MouseMotion. Pour finir, Le fichier Configuration affiche à l'écran le dessin en prenant en compte a chaque les évènements réalisés par l'utilisateur.

Fichier Main.py :
Il définit et utilise des configurations.

2. Question (1) b

Configuration({ 'screenPosition' : -5, 'xAxisColor': [1, 1, 0] }).display()
Cette modification permet de modifier la couleur de l'axe x en jaune. Voici la fenêtre d'affichage :

Le chaînage de l'appel des méthodes setParameter() et display() est possible car le résultat de display est objet Configuration et la méthode setParameter() s'applique sur un objet Configuration.

Un traitement particulier est effectué dans le « setter » pour le paramètre screenPosition. Ce traitement particulier doit être effectué car pour utiliser OpenGL, il faut définir la matrice de perspective. C'est donc pour cela que dans le setter, lorsqu'on veut initialiser le paramètre associé à 'screen position' il faut d'abord passer la configuration dans la méthode initializeTransforationMatrix afin de pouvoir afficher le résultat dans OpenGL.

3. Question (1) c

Voici l'instruction que nous avons ajouté (à la fin) ainsi que l'affichage :

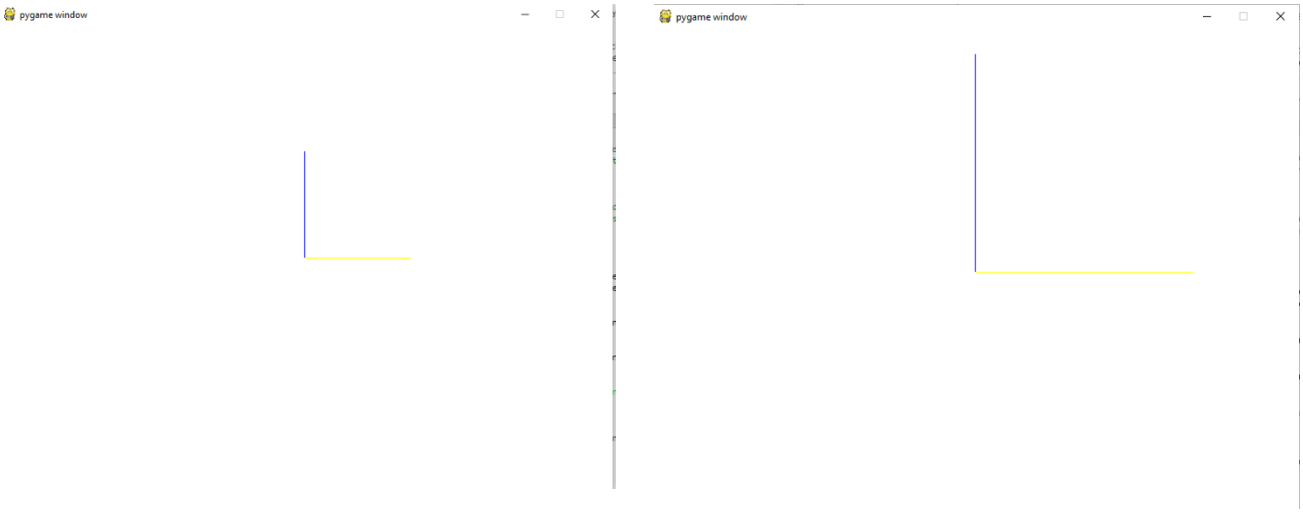
```
def initializeTransformationMatrix(self):
    gl.glMatrixMode(gl.GL_PROJECTION)
    gl.glLoadIdentity()
    glu.gluPerspective(70, (self.screen.get_width()/self.screen.get_height()), 0.1, 100.0)
    gl.glMatrixMode(gl.GL_MODELVIEW)
    gl.glLoadIdentity()
    gl.glTranslatef(0.0,0.0, self.parameters['screenPosition'])
    gl.glRotatef(-90, 1, 0, 0)
```

V. Mise en place des interactions avec l'utilisateur avec Pygame

1. Question (1) d

Nous avons ajouté des conditions sur les évènements. Si l'évènement réalisé (évènement entré en paramètre) correspond à l'évènement Page Up ou Page Down du dictionnaire, alors on fait un zoom ou un dezoome. Pour cela nous avons utilisé les fonctions glScalef en changeant les paramètres selon le cas (zoom ou dezoom).

```
def processKeyDownEvent(self):
    # Rotates around the z-axis
    if self.event.dict['unicode'] == 'Z' or (self.event.mod & pygame.KMOD_SHIFT and self.event.key == pygame.K_z):
        gl.glRotate(-2.5, 0, 0, 1)
    elif self.event.dict['unicode'] == 'z' or self.event.key == pygame.K_z:
        gl.glRotate(2.5, 0, 0, 1)

    # Draws or suppresses the reference frame
    elif self.event.dict['unicode'] == 'a' or self.event.key == pygame.K_a:
        self.parameters['axes'] = not self.parameters['axes']
        pygame.time.wait(300)

    elif self.event.dict['unicode'] == 'PageUp' or self.event.key == pygame.K_PAGEUP:
        gl.glScalef(1.1, 1.1, 1.1)

    elif self.event.dict['unicode'] == 'PageDown' or self.event.key == pygame.K_PAGEDOWN:
        gl.glScalef(1/1.1, 1/1.1, 1/1.1)
```



2. Question (1) e

Le sens de l'action sur la mollette est fourni par l'attribut button, associé à l'événement, qui prend les valeurs 4 ou 5. 4 correspond au dezoom et 5 au zoom.

```
# Processes the MOUSEBUTTONDOWN event
def processMouseButtonDownEvent(self):
    if self.event.button == 4:
        gl.glScalef(1/1.1, 1/1.1, 1/1.1)
    elif self.event.button == 5:
        gl.glScalef(1.1, 1.1, 1.1)
```

3. Question (1) f

```
def processMouseMotionEvent(self):
    if self.event.type == pygame.MOUSEMOTION:
        if pygame.mouse.get_pressed()[0]==1:
            gl.glRotate(self.event.rel[1], -1, 0, 0)
            gl.glRotate(self.event.rel[0], 0, 0, -1)
        else:
            gl.glRotate(0,0,0,0)
        if pygame.mouse.get_pressed()[2] == 1:
            gl.glTranslate(0.1*self.event.rel[0], 0, 0.1*self.event.rel[1])
        else:
            gl.glTranslate(0,0,0)
```

VI. Création d'une section

1. Question 2(a)

Dans cette méthode on a d'abord défini les sommets 4 sommets manquants (4 sommets de la face du fond) puis les 5 faces manquantes du parallélépipède.

```
def generate(self):
    self.vertices = [
            [0, 0, 0 ],
            [0, 0, self.parameters['height']],
            [self.parameters['width'], 0, self.parameters['height']],
            [self.parameters['width'], 0, 0],
            [0, self.parameters['thickness'],0],
            [0, self.parameters['thickness'],self.parameters['height']],
            [self.parameters['width'], self.parameters['thickness'],self.parameters['height']],
            [self.parameters['width'], self.parameters['thickness'],0]

            ]
    self.faces = [
            [0, 3, 2, 1],
            [1, 2, 6, 5],
            [4, 7, 6, 5],
            [0, 3, 7, 4],
            [0, 4, 5, 1],
            [3, 7, 6, 2],
            ]
```

2. Question 2(b)

La fonction Q2b () crée une configuration et donc l'espace nécessaire afin de créer la face. Ensuite on ajoute une section en lui donnant une position ainsi qu'une dimension. Enfin, l'extension display permet d'afficher la section.

On a d'abord fait ce code, qui trace seulement une seule face en grise.

```
def draw(self):


    gl.glPushMatrix()
    gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
    gl.glRotate(self.parameters['orientation'],0,0,1)
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)

    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
    gl.glVertex3fv([0, 0, 0])
    gl.glVertex3fv([1, 0, 0])
    gl.glVertex3fv([1, 0, 1])
    gl.glVertex3fv([0, 0, 1])
    gl.glEnd()
    gl.glPopMatrix()
```

On a donc mis en place une boucle for pour tracer les 6 faces du parallélépipède.
Voici notre code :

```
def draw(self):


    gl.glPushMatrix()
    gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
    gl.glRotate(self.parameters['orientation'],0,0,1)
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
    for f in self.faces:
        gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
        gl.glColor3fv(self.parameters['color']) # Couleur gris moyen
        for g in f:
            gl.glVertex3fv(self.vertices[g])
        gl.glEnd()
    gl.glPopMatrix()
```

Voici ce que nous obtenons : (nous obtenons bien un parallélépipède qui a 6 faces)

3. Question 2(c)

Pour assombrir la couleur, on a choisi un facteur de 0.3 (inférieur à 1)

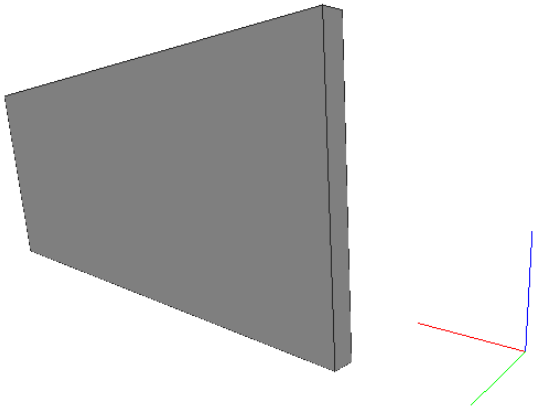
Voici notre code pour la méthode draw edges qui nous permet de mettre en évidence les arrêtes du parallélépipède :

```
def drawEdges(self):
    gl.glPushMatrix()
    gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
    gl.glRotate(self.parameters['orientation'],0,0,1)
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE)
    for i in self.faces:
        gl.glBegin(gl.GL_QUADS)
        gl.glColor3fv([self.parameters['color'][0]*0.3,self.parameters['color'][1]*0.3,self.parameters['color'][2]*0.3])
        for j in i:
            gl.glVertex3fv(self.vertices[j])
        gl.glEnd()
    gl.glPopMatrix()
```

Voici comment nous avons modifier la méthode draw :
En plaçant un ‘if’ avant notre code précédent, cela permet d’exécuté la méthode draw edges en premier quand le paramètre edges prend la valeur true.

```
# Draws the faces
def draw(self):
    if self.parameters['edges']:
        self.drawEdges()

    gl.glPushMatrix()
    gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
    gl.glRotate(self.parameters['orientation'],0,0,1)
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
    for f in self.faces:
        gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
        gl.glColor3fv(self.parameters['color']) # Couleur gris moyen
        for g in f:
            gl.glVertex3fv(self.vertices[g])
        gl.glEnd()
    gl.glPopMatrix()
```

Voici ce que nous obtenons : (les arrêtes sont bien représentés en plus foncé)



VII. Création des murs

1. Question 3(a)

Dans le fichier Wall et notamment dans son constructeur, il y a une succession de tests qui vérifient si les paramètres de parameters sont bien présents. S'il en manque, il va en fixer par « défaut » avec des valeurs simples (il les initialise).

On crée également une liste d'objets (dans laquelle nos objets sont de type fenêtre, portes, etc). Ces objets sont associés à une section selon des paramètres donnés et requis dans la classe section. Avec la commande « .append », nous ajoutons les objets créés à la liste des objets dans l'objectif de pouvoir les représenter plus tard.

Voici notre code :
On parcourt la liste d'objet et on dessine au fur et à mesure tous les objets de la liste dans notre environnement.

```
# Draws the faces
def draw(self):
    gl.glPushMatrix()
    gl.glRotate(self.parameters['orientation'],0,0,1)
    for k in self.objects:
        k.draw()
    gl.glPopMatrix()
```

Fonction Q3a :

```
def Q3a():
    return Configuration().add(
            Wall({'position': [1, 1, 0], 'width':7, 'height':2.6, 'edges': True})
            ).display()
```

VIII. Création d'une maison

Voici notre code :
Comme indiqué dans l'énoncé, nous n'avons pas gérer la translation car elle est déjà gérée dans la classe section.

```
# Draws the house
def draw(self):
    gl.glPushMatrix()
    gl.glRotate(self.parameters['orientation'],0,0,1)
    for k in self.objects:
        k.draw()
    gl.glPopMatrix()
```

Ce code permet de coller 4 murs en choisissant leur orientation et leur coordonnée :

```
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({'position': [0, 0, 0], 'width':5, 'height':2.6, 'edges': True, 'orientation':0} )
    wall2 = Wall({'position': [0, 0, 0], 'width':5, 'height':2.6, 'edges': True, 'orientation':90})
    wall3 = Wall({'position': [0, 5, 0], 'width':5, 'height':2.6, 'edges': True,'orientation':0})
    wall4 = Wall({'position': [0, -5, 0], 'width':5, 'height':2.6,'edges': True,'orientation':90})
    house = House()
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    return Configuration().add(house)
```

Et voici ce que nous avons obtenus :

IX. Conclusion

Nous n'avons pas réussi à finir ce TP. Nous avons principalement rencontré des difficultés pour la mise en place des interactions avec l'utilisateur avec Pygame. Globalement, nous avons plus apprécié la deuxième partie de ce TP. Nous avons appris à gérer des objets. Ce Tp nous a permis de nous familiariser avec les objets, Pygame et OpenGl.